**（三）文档注释**

使用/\*\* \*/进行注释：

/\*\*

阿平也太帅了吧

\*/

文档注释主要是用来生成java开发文档javadoc的，生成的开发文档和Java本身的API帮助文档是一样的，也就是对你所写的类进行解释说明，所以它还需要搭配一些文档标记，进行解释说明，而且在文档注释中可以使用HTML语言，jdk源码中有大量的文档注释，所以弄懂文档注释可以帮助你更好的看懂源码。

文档注释通常还会配合HTML标签进行使用，比较常用的标签有<p><pre>等标签，p标签用于表示段落，pre标签可用于显示计算机源码。

**注意**：pre标签中如果有小于号、大于号、例如泛型 在生产javadoc时会报错。

**1、文档标记**

**（1）通用的文档标记**

以下文档标记在类、方法、变量和常量上都经常使用。

1. @link： 用于快速链接到相关代码，使用格式：{@link 包名.类名#方法名(参数类型)}  
   *// 完全限定的类名*  
   {@link java.util.Collections}  
     
   *// 省略包名，只写类名*  
   {@link String}  
     
   *// 省略类名，表示指向当前的某一个方法*  
   {@link #toString()}  
     
   *// 完全限定方法名，包名.类名.方法名(参数类型)*  
   {@link java.lang.String#charAt(int)}
2. @code： 将文本标记为代码样式文本，一般在Javadoc中只要涉及到类名或者方法名，都需要使用@code进行标记，使用格式：{@code text}，其会被解析为 text  
   *//标记类名*  
   {@code ArrayList}  
     
   *//标记方法名*  
   {@code isEmpty}  
     
   *//标记某个代码关键字*  
   {@code null}

**（2）类上常用文档标记**

1. @param：如果一个类支持泛型时，可以通过@param来解释泛型的类型  
   */\*\**  
   *@param <E> the type of elements in this list*  
   *\*/*
2. @author：用来标记作者，如果一段程序是由多个作者来维护，则可以标记多个@author,@author 后面可以跟作者姓名(也可以附带作者邮箱地址)、组织名称(也可以附带组织官网地址)  
   *// 纯文本作者*  
   @author Rod Johnson  
     
   *// 纯文本作者，邮件*  
   @author Igor Hersht, igorh@ca.ibm.com  
     
   *// 超链接邮件 纯文本作者*  
   @author <a href="mailto:ovidiu@cup.hp.com">Ovidiu Predescu</a>  
     
   *// 纯文本邮件*  
   @author shane\_curcuru@us.ibm.com  
     
   *// 纯文本 组织*  
   @author Apache Software Foundation  
     
   *// 超链接组织地址 纯文本组织*  
   @author <a href="[https://jakarta.apache.org/turbine](https://link.zhihu.com/?target=https%3A//jakarta.apache.org/turbine)"> Apache Jakarta Turbine</a>
3. @see :另请参阅的意思，一般用于标记与本类相关联的类，该标注可以用在类或方法上。  
   */\*\**  
   *\* @see IntStream*  
   *\* @see LongStream*  
   *\* @see DoubleStream*  
   *\* @see <a href="package-summary.html">java.util.stream</a>*  
   *\* /*
4. @since：表示从以下版本开始有这个类，标记文件创建时项目当时对应的版本，后面可以跟版本号或是时间。  
   *//跟版本号，以下是ArrayList类里的标记，说明从jdk1.2开始就有该类了*  
   */\**  
   *\* @since 1.2*  
   *\*\*/*  
   *//跟时间*  
   */\*\**  
   *\* @since 20 April 2021*  
   *\*/*
5. @version：用于标记当前类版本，默认为1.0  
   */\*\**  
   *\* @version 1.0*  
   *\*/*

以上是类上常用的文档标注，类上的文档格式如下：

1. 概要描述：通常用一段话简要的描述该类的基本内容。
2. 详细描述：通常用几大段话详细描述该类的功能与相关情况。
3. 文档标注：用于标注该类的作者、时间、版本、参略等信息。

以下是String类的中文档标注的事例：

/\*\*

\* The {@code String} class represents character strings. All

\* string literals in Java programs, such as {@code "abc"}, are

\* implemented as instances of this class.

\* <p>

\* Strings are constant; their values cannot be changed after they

\* are created. String buffers support mutable strings.

\* Because String objects are immutable they can be shared. For example:

\* <blockquote><pre>

\* String str = "abc";

\* </pre></blockquote><p>

\* is equivalent to:

\* <blockquote><pre>

\* char data[] = {'a', 'b', 'c'};

\* String str = new String(data);

\* </pre></blockquote><p>

\* Here are some more examples of how strings can be used:

\* <blockquote><pre>

\* System.out.println("abc");

\* String cde = "cde";

\* System.out.println("abc" + cde);

\* String c = "abc".substring(2,3);

\* String d = cde.substring(1, 2);

\* </pre></blockquote>

\* <p>

\* The class {@code String} includes methods for examining

\* individual characters of the sequence, for comparing strings, for

\* searching strings, for extracting substrings, and for creating a

\* copy of a string with all characters translated to uppercase or to

\* lowercase. Case mapping is based on the Unicode Standard version

\* specified by the {@link java.lang.Character Character} class.

\* <p>

\* The Java language provides special support for the string

\* concatenation operator (&nbsp;+&nbsp;), and for conversion of

\* other objects to strings. For additional information on string

\* concatenation and conversion, see <i>The Java&trade; Language Specification</i>.

\*

\* <p> Unless otherwise noted, passing a {@code null} argument to a constructor

\* or method in this class will cause a {@link NullPointerException} to be

\* thrown.

\*

\* <p>A {@code String} represents a string in the UTF-16 format

\* in which <em>supplementary characters</em> are represented by <em>surrogate

\* pairs</em> (see the section <a href="Character.html#unicode">Unicode

\* Character Representations</a> in the {@code Character} class for

\* more information).

\* Index values refer to {@code char} code units, so a supplementary

\* character uses two positions in a {@code String}.

\* <p>The {@code String} class provides methods for dealing with

\* Unicode code points (i.e., characters), in addition to those for

\* dealing with Unicode code units (i.e., {@code char} values).

\*

\* <p>Unless otherwise noted, methods for comparing Strings do not take locale

\* into account. The {@link java.text.Collator} class provides methods for

\* finer-grain, locale-sensitive String comparison.

\*

\* @implNote The implementation of the string concatenation operator is left to

\* the discretion of a Java compiler, as long as the compiler ultimately conforms

\* to <i>The Java&trade; Language Specification</i>. For example, the {@code javac} compiler

\* may implement the operator with {@code StringBuffer}, {@code StringBuilder},

\* or {@code java.lang.invoke.StringConcatFactory} depending on the JDK version. The

\* implementation of string conversion is typically through the method {@code toString},

\* defined by {@code Object} and inherited by all classes in Java.

\*

\* @author Lee Boynton

\* @author Arthur van Hoff

\* @author Martin Buchholz

\* @author Ulf Zibis

\* @see java.lang.Object#toString()

\* @see java.lang.StringBuffer

\* @see java.lang.StringBuilder

\* @see java.nio.charset.Charset

\* @since 1.0

\* @jls 15.18.1 String Concatenation Operator +

\*/

public final class String

implements java.io.Serializable, Comparable<String>, CharSequence {

}

**（3）方法上常用文档标记**

1. @param：该文档标记后面写方法的参数名，再写参数描述。  
   */\*\**  
   *\* @param str*  
   *\* the {@code CharSequence} to check (may be {@code null})*  
   *\*/*  
   public static boolean containsWhitespace(@Nullable CharSequence str) {}
2. @return：该文档标记后面写返回值得描述。  
   */\*\**  
   *\* @return {@code true} if the {@code String} is not {@code null}, its*  
   *\*/*  
   public static boolean hasText(@Nullable String str){}
3. @throws：该文档标记后面写异常的类型和异常的描述，用于描述该方法可能抛出的异常。  
   */\*\**  
   *\* @throws IllegalArgumentException when the given source contains invalid encoded sequences*  
   *\*/*  
   public static String uriDecode(String source, Charset charset){}
4. @exception：该标注用于描述方法签名throws对应的异常。  
   */\*\**  
   *\* @exception IllegalArgumentException if <code>key</code> is null.*  
   *\*/*  
   public static Object get(String key) throws IllegalArgumentException {}
5. @see：可用在类与方法上，表示参考的类或方法。  
   */\*\**  
   *\* @see java.net.URLDecoder#decode(String, String)*  
   *\*/*  
   public static String uriDecode(String source, Charset charset){}

以上是方法上常用的文档标注，方法上的文档格式如下：

1. 概要描述：通常用一段话简要的描述该方法的基本内容。
2. 详细描述：通常用几大段话详细描述该方法的功能与相关情况。
3. 文档标注：用于标注该方法的参数、返回值、异常、参略等信息。

以下是String类中charAt方法的示例：

/\*\*

\* Returns the {@code char} value at the

\* specified index. An index ranges from {@code 0} to

\* {@code length() - 1}. The first {@code char} value of the sequence

\* is at index {@code 0}, the next at index {@code 1},

\* and so on, as for array indexing.

\*

\* <p>If the {@code char} value specified by the index is a

\* <a href="Character.html#unicode">surrogate</a>, the surrogate

\* value is returned.

\*

\* @param index the index of the {@code char} value.

\* @return the {@code char} value at the specified index of this string.

\* The first {@code char} value is at index {@code 0}.

\* @exception IndexOutOfBoundsException if the {@code index}

\* argument is negative or not less than the length of this

\* string.

\*/

public char charAt(int index) {}

**（4）变量和常量上的文档规范**

变量和常量上用的比较多的文档标记是@link和@code，主要注释该常量或变量的基本用法和相关内容。

以下是示例：

/\*\*

\* The value is used for character storage.

\*

\* @implNote This field is trusted by the VM, and is a subject to

\* constant folding if String instance is constant. Overwriting this

\* field after construction will cause problems.

\*

\* Additionally, it is marked with {@link Stable} to trust the contents

\* of the array. No other facility in JDK provides this functionality (yet).

\* {@link Stable} is safe here, because value is never null.

\*/

private final byte[] value;